Schriftliche Ausarbeitung für das Seminar Softwareentwicklung in der Wissenschaft Thema: Machine Learning mit TensorFlow in den Geowissenschaften

Stefan Knispel
Matrikelnummer: 665674
Betreuer: Tobias Sebastian Finn

3. August 2018

Inhaltsverzeichnis

1 Motivation

Die Anwendungsgebiete von Machine Learning erstrecken sich über sehr viele verschiedene Disziplinen. Andrew Ng ist ein sehr bekannter Forscher im Bereich künstliche Intelligenz. Er sagte 2016: „If a typical person can do a mental task with less than one second of thought, we can probably automate it using AI either now or in the near future.“ Ein Jahr später wurde er von Ingo Miersw, Gründer von RapidMiner, verbessert: „It is safe to go already with a minute instead of a second.“ An diesen Zitaten kann man leicht erkennen, dass Machine Learning bei sehr vielen Problemen benutzt werden kann. Weiterhin zeigt sich, dass Machine Learning eines der aktuellsten Themen in der Computerwissenschaft ist. Innerhalb nur eines Jahres kann hier durch die intensive Forschung sehr viel an Fortschritt geschehen. Ein Zitat von Ahmet Taspinar, ein Datenwissenschaftler, hat mich während meiner Masterarbeit sehr fasziniert: „The use cases for using Convolution Neural Networks are really only limited by

your imagination.“ Damit Machine Learning in vielen Anwendungsbereichen genutzt werden kann, hat Google eine Programmierschnittstelle produziert, welche öffentlich zugänglich gemacht und darauf geachtet wurde, eine einfache Handhabung zu ermöglichen! Sie heißt TensorFlow. Mit TensorFlow versuche ich ebenfalls innerhalb meines Masterprogramms ein Problem aus der angewandten Seismik (Geophysik) zu lösen.

2 Machine Learning - kurze Einführung

Machine Learning ist ein Bereich in der Computerwissenschaft, welcher statistische Techniken verwendet um dem Computer die Möglichkeit zu geben, mit Daten zu lernen, ohne direkt dafür programmiert zu werden. Es ist eine Anwendung von künstlicher Intelligenz, wobei Neuronale Netze ein Teil davon sind. Diese sind dem menschlichen Gehirn nachempfunden, um Muster zu erkennen. Im weiteren werden Neuronale Netze genauer erklärt, da dies hier das Handwerkszeug ist, um einen Machine Learning Algorithmus umzusetzen. Neuronale Netze sind als Graphen mit mehreren Schichten aufgebaut. Bei der Verwendung von vielen Schichten spricht man dann von Deep Learning. Es werden nicht-lineare Modelle an vorhandene Daten angepasst und mittels supervised oder unsupervised Learning verbessert. Das Ziel ist eine vereinfachte Darstellung der Rohdaten um Muster zu erkennen (low-level features oder higher-level features). Low-level features könnten zum Beispiel die generelle Form eines Autos darstellen. Higher-level features zum Beispiel die Existenz eines Seitenspiegels.
Neuronen sind nicht-lineare Transformationen

$$y = f(w^T x + b)$$

. Es wird quasi eine lineare Regression vollzogen, wobei w^T anzupassende Gewichte und b einen anzupassenden Bias darstellen. Das Neuron wird danach aktiviert mit einer nicht-linearen Funktion $f()$ um eine nicht-lineare Regression zu erhalten. Die Regression kann bei mehreren Variablen in einem höherdimensionalen Raum durchgeführt werden. Mit einer Backpropagation werden die anfangs zufällig verteilten Variablen (Gewichte und Bias) mit jedem Trainingsdurchlauf verbessert, um die Verlustfunktion minimal zu bekommen. So wird die optimale Regressionsgerade angepasst.
Dabei wird mit einem Satz von Daten und im supervised Learning Fall mit den dazugehörigen Labeln trainiert. Mit einem Validierungs-Datensatz kann getestet werden, wie gut der Graph und damit die Regressionsgerade angepasst wurde. Mit einer Verlust-Funktion wird die Performance getestet. Die Backpropagation und damit das Verbessern der Variablen und Biases wird mit einer Gradientenberechnung durchgeführt (GradientDescent ist hier der Standard). Es wird der Gradient der Verlust-Funktion berechnet und im simplen Fall mit einer Lernrate multipliziert. Das Ergebnis wird dann von den anzupassenden Variablen subtrahiert. Wenn die Verlust-Funktion auf einem ausreichend geringen Niveau verharrt, ist das Training beendet.
Neuronale Netzwerke werden für Klassifikationen, Regressionsprobleme, Objektenerkennung, Spracherkennung, Übersetzung und vieles weitere angewendet.

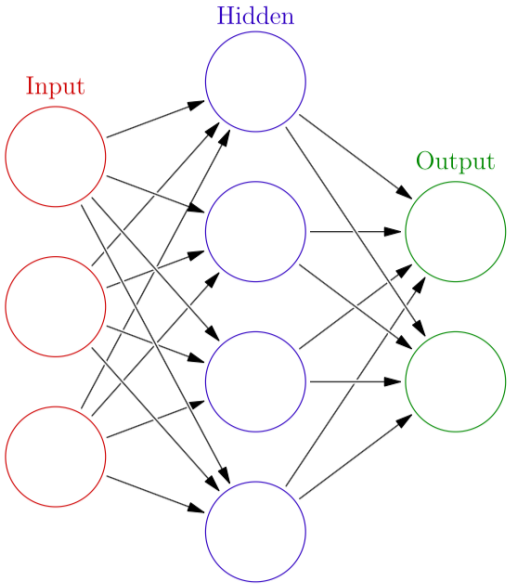

Abbildung 1: Neuronales Netzwerk mit 2 Schichten [Gra16]

3 Tensorflow - Großskaliges Machine Learning auf heterogen verteilten Systemen [MA15]

TensorFlow ist ein Interface um Machine Learning Algorithmen auszudrücken und um diese zu implementieren und auszuführen. Es ist für eine Vielzahl an heterogenen Systemen (Smartphones, Tablets bis hin zu großskaligen Systemen mit 1000 von GPU Karten) nutzbar. Noch dazu sind viele Algorithmen eingeschlossen (z.B. Deep Neural Networks). TensorFlow wurde von Google Brain seit 2011 entwickelt, zuerst hauptsächlich für Google Produkte, jedoch 2015 als Open-Source für die Forschung und für Produktion zur Verfügung gestellt, um Machine Learning für viele Anwendungsbereiche zugänglich zu machen. Anwendungsbeispiele sind Computer Vision, Robotics, Übersetzung, Geografische Anwendungen und vieles mehr. In meiner Masterarbeit nutze ich TensorFlow für ein geophysikalisches Problem, welches ich am Ende näher beschreiben werde.
Vor TensorFlow benutzte Google DistBelief (auch für unsupervised learning, Spracherkennung, Google Suche, personalisierte Werbung, Google Maps etc.), welches auch für das Go-Spiel aus China genutzt wurde, um professionelle menschliche Spieler in dem Spiel zu besiegen. Mit der Erfahrung aus DistBelief wurde TensorFlow entwickelt, wobei viel Wert auf Flexibilität, Schnelligkeit und Einfachheit für Experimente in der Forschung gelegt wurde, aber auch eine starke Performance und ein robuster Produktionsbetrieb waren sehr wichtig. TensorFlow ist schneller als DistBelief, die Performance ist besser, es unterstützt mehr Modelle und kann auf einer größeren Anzahl an Systemen angewendet werden.

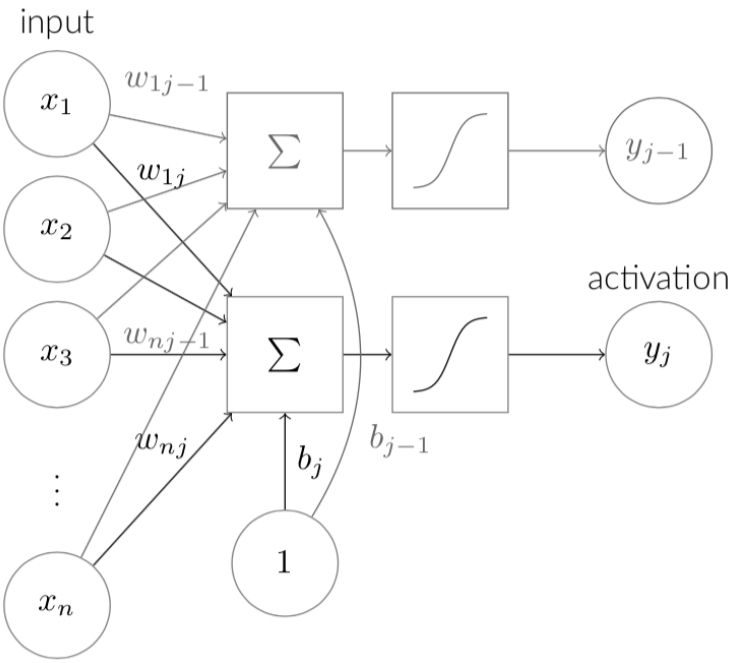

Abbildung 2: Neuronales Netzwerk [Gra16]

3.1 Programming Model and Basic Concepts

Eine TensorFlow Berechnung beruht auf einem gerichteten Graphen, welcher Daten transportiert. Der Graph besteht aus einem Set von Knoten, welche jeweils Berechnungen ausführen. Mit Python oder C++ können Graphen erzeugt werden. Ein Beispiel folgt in Abbildung 3 und 4. Generell hat jeder Knoten keinen oder mehrere Inputs und keinen oder mehrere Outputs. Jeder Knoten repräsentiert eine Operation, welche Objekte erzeugt (Tensoren etc.). Werte, welche über „normal edges“ fließen (Output zu Input) sind Tensoren (mehrdimensionale Arrays), wohingegen „special edges“ „controll dependencies“ genannt werden. Hier fließen keine Daten. „Bei controll dependencies“ muss der Quellcode für den Knoten erst beendet werden, bevor der Ziel-Knoten anfängt ausgeführt zu werden. TensorFlow Implementation fügt manchmal automatisch „controll dependencies“ ein.

3.2 Operationen und Kernels

Eine Operation repräsentiert eine Berechnung (Matrix-Multiplikation, Addition etc.), wobei diese auch Attribute haben können. Diese Attribute werden häufig verwendet, um Operationen polymorph zu machen (an unterschiedlichen Datentypen anwendbar). Ein Kernel ist eine Implementation einer Operation, welche auf einem bestimmten Gerätetyp ausgeführt werden soll (CPU, GPU). Ein TensorFlow Programm ist ein Set von Operationen und Kernels, welche erweitert werden können, um den Graph zu vergrößern.

```
import tensorflow as tf

b = tf.Variable(tf.zeros([100]))                    # 100-d vector, init to zeroes
W = tf.Variable(tf.random_uniform([784,100],-1,1)) # 784x100 matrix w/rnd vals
x = tf.placeholder(name="x")                        # Placeholder for input
relu = tf.nn.relu(tf.matmul(W, x) + b)              # Relu(Wx+b)
C = [...]                                           # Cost computed as a function
                                                    # of Relu

s = tf.Session()
for step in xrange(0, 10):
  input = ...construct 100-D input array ...        # Create 100-d vector for input
  result = s.run(C, feed_dict={x: input})           # Fetch cost, feeding x=input
  print step, result
```

Abbildung 3: Tensorflow-Code Beispiel [MA15]

3.3 Sessions

Der Anwender interagiert mit dem TensorFlow-System mit dem Kreieren einer Session. Die default-Session ist leer (aktueller Graph muss erweitert werden) und mit „run“ wird die Session ausgeführt. Die TensorFlow Implementation berechnet dann die transitive Hülle (Erweiterung einer direkten Relation, welche alle indirekten Relationen enthält) aller Knoten, welche für den erforderten Output verlangt wwird und führt diese aus. Meistens wird der ganze Graph ausgeführt, und das mehrere tausend bis millionen Male.

3.4 Variablen

Tensoren überdauern keine ganze Ausführung eines Graphen. Variablen geben daher Zugriff auf einen beständigen Tensor, welcher über mehrere Ausführungen eines Graphen verändert werden kann. Die Variable wird als Teil von „run“ im Training geupdatet. Hier wird die lineare Regression durchgeführt und somit die Variablen der Gewichte und des Bias verbessert. Um nicht-lineare Beziehungen zu ermöglichen, wird eine Aktivierungsfunktion angewendet (hier ReLU). Die ReLu Aktivierungsfunktion ist null für Werte kleiner null und linear mit der Steigung 1 für Werte größer als null.

3.5 Implementation

Die Hauptkomponente in TensorFlow-Systemen ist der Benutzer. Dieser nutzt das Session Interface um mit dem Master-Prozess zu kommunizieren. Der Master-Prozess kommuniziert dann mit den Arbeiter-Prozessen (jeder davon hat Zugang zu einem oder mehreren CPU- oder GPU- Karten). Dabei gibt es eine lokale und eine verteile Implementation (Abbildung 5).
Bei der lokalen Implementation befinden sich Master und Arbeiter auf dem gleichem System, wobei die Knoten nach Relation ausgeführt werden. Für einen Knoten wird die Anzahl der Knoten berechnet, welche in seiner Relation stehen, jedoch noch nicht ausgeführt wurden. Ist diese Anzahl gleich null, kann der Knoten ausgeführt werden

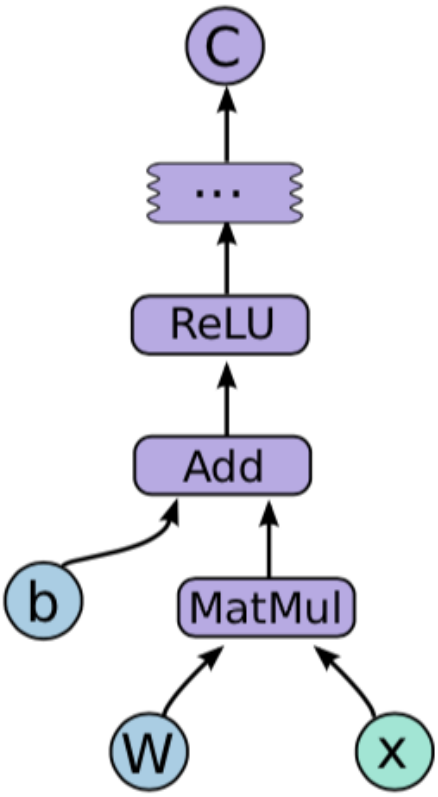

Abbildung 4: Zugehöriger Graph zu Abbildung 3 [MA15]

bzw. wird in eine Warteschlange eingereiht.
Bei der verteilten Implementation befinden sich Master und Arbeiter auf unterschiedlichen Systemen, teilen dennoch den größten Code-Teil mit der lokalen Implementation. Hier treten zwei Probleme auf: Welcher Knoten befindet sich auf welchem Gerät? Dazu gibt es einen extra Algorithmus: Jeder Knoten wird nacheinander auf alle möglichen Geräte gelegt und mit einem simulierten Graphdurchlauf wird die Berechnungszeit abgeschätzt (heuristisch). Die Verteilung mit der kürzesten Berechnungszeit wird am Ende gewählt, um die Knoten sinnvoll zu verteilen. Jedoch ist das noch ein Feld auf dem geforscht wird. Ein weiteres Problem ist die Kommunikation zwischen den Knoten, welche geregelt werden muss. Dafür werden Send/Receive Knoten in jeden Subgraph eingesetzt. Für jedes Gerät wird nur ein Send/Receive Knoten verwendet, um die Kommunikation zu kanalisieren (Abbildung 6). Die Reihenfolge der Jobs wird mit einem Cluster-Scheduling-System geregelt.

3.6 Fehlertoleranz

Bei Fehlern bricht die Berechnung des Graphen ab und startet komplett neu. Für die Sicherheit der Daten und um einen großen Zeitverlust zu vermeiden, ist jede Variable mit einem Sicherungs-Knoten verbunden, welcher periodisch ausgeführt wird.

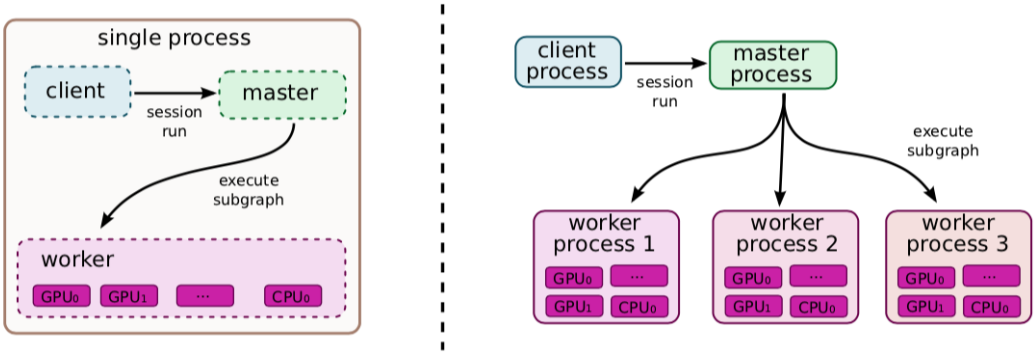

Abbildung 5: Einzelmaschine und verteilte System-Struktur [MA15]

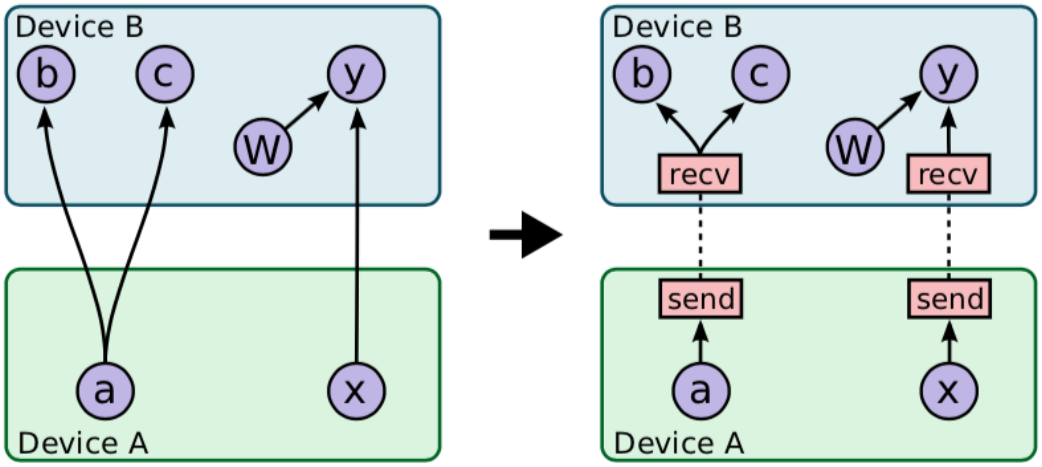

Abbildung 6: Einfügen von Send/Receive Knoten [MA15]

Zusätzlich sind sie mit einem Restore-Knoten verbunden, welcher bei der ersten Iteration nach dem Neustart ausgeführt wird.

3.7 Training

Das Training und damit die Optimierung wird meistens über eine Gradientenberechnung der Kostenfunktion durchgeführt. Der Gradient wird anhand von Abbildung 7 berechnet. Für den Gradienten von zum Beispiel Tensor C in Abhängigkeit von Tensor I wird zuerst der Pfad von I nach C gesucht. Danach wird dieser Pfad rückwärts abgelaufen und pro Operation wird ein Knoten eingefügt, welcher den partiellen Gradienten für jene Operation beinhaltet. Mit diesem Verfahren kann dann der Gradient von Tensor C berechnet werden. In TensorFlow gibt es dafür eine built-in Unterstützung, welche automatisch den Gradienten berechnet und damit die Variablen updatet.

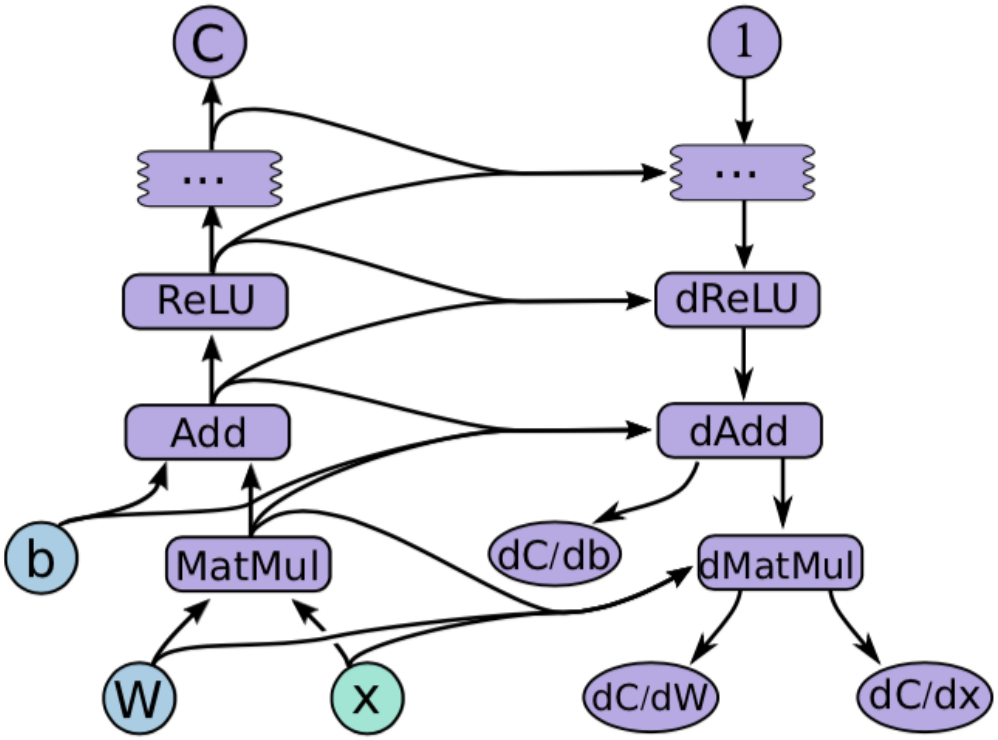

Abbildung 7: Gradientenberechnung für den Graph in Abbildung 4 [MA15]

3.8 Partielle Ausführung

Eine weitere Besonderheit von TensorFlow ist die Unterstützung der Ausführung einzelner Subgraphen. Daten können an jedem Knoten reingegeben werden und an jedem Knoten können Daten ausgelesen werden. Dafür werden feed and fetch-Knoten angehängt und nur der notwendige Teil des Graphens berechnet (Abbildung 8).

3.9 Eingabe Operationen

Durch die Dateneingabe mit speziellen Eingabe-Knoten, welche Tensoren enthalten, die aus einem oder mehreren Beispielen aus dem Datensatz bestehen, ist es möglich die Daten direkt aus dem Speichersystem in den Speicher der Maschine zu laden (Speichersystem – Arbeiter vs. Speichersystem – Client – Arbeiter).

3.10 Queues

Meistens wird die normale First-in-First-Out-Queue verwendet, aber zusätzlich gibt es noch die Shuffeling-Queue. Sie hilft dem Machine Learning Algorithmus die Prozessordnung zufällig zu verteilen.

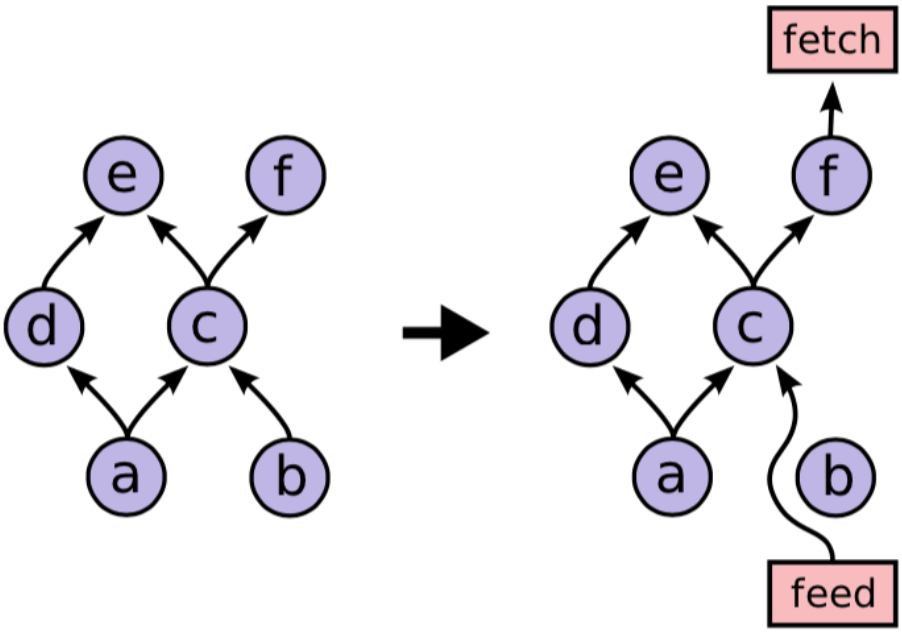

Abbildung 8: Partielle Ausführung des Graphens [MA15]

3.11 Tipps von Google Brain Team

Während der Entwicklung und während der Arbeit mit TensorFlow hat das Google Brain Team einige Tipps notiert, welche bei einer eigenen Verwendung beachtet werden sollten. Zum einen sollte man Tools erstellen, um die genaue Anzahl an Parametern in einem Modell zu sehen. Diese demonstrieren kleine Fehler im Neuronalen Netz. Weiterhin sollte man immer erst mit einem kleinen Netzwerk anfangen und es danach vergrößern, da es somit einfacher fällt Fehler zu finden. Um zu testen, ob die Verlustfunktion funktioniert, kann man die Lernrate auf null stellen. Die Variablen dürften sich nicht verändern. Noch dazu sollte man immer zuerst mit einer lokaler Implementierung testen, da die Verteilte schwieriger ist. Wichtig sind außerdem das Beachten von numerischen Fehlern, da Machine learning Algorithmen anfällig sind für numerische Instabilität. Nicht-finite floating points sollten vermieden werden.

3.12 TensorBoard (Visualisierung)

Die Visualisierung eines Graphen kann unter Umständen extrem komplex werden. Graphen bestehen nicht selten aus mehreren tausend bis millionen Knoten. Mit dem built-in Tool TensorBoard soll die Visualisierung sehr einfach gestaltet werden. Dazu werden Knoten in Blöcke identischer Strukturen unterteilt. Die Visualisierung ist interaktiv, es ist möglich zu zoomen und zu verschieben und weiterhin können gruppierte Knoten angezeigt und diese gegebenenfalls auch ausgeblendet werden. Die Änderung der Variablen etc. ist über jede Iteration darstellbar, aber auch über die Berechnungszeit und über die Uhrzeit. Es gibt eine große Variation an Darstellungen,

unter anderem skalare Zusammenfassungen, Histogramme und eine Darstellung mit Bildern.
Um TensorBoard nutzen zu können müssen zusätzlich zum Graphen noch weitere zeilen Code programmiert werden, welche die gewünschten Daten als Zusammenfassung in einer log-Datei speichern. Diese log-Datei kann dann mit einem Browser geöffnet werden. Ein Beispiel der Visualisierung findet sich in Abbildung 9. Der Graph ist sehr komplex.

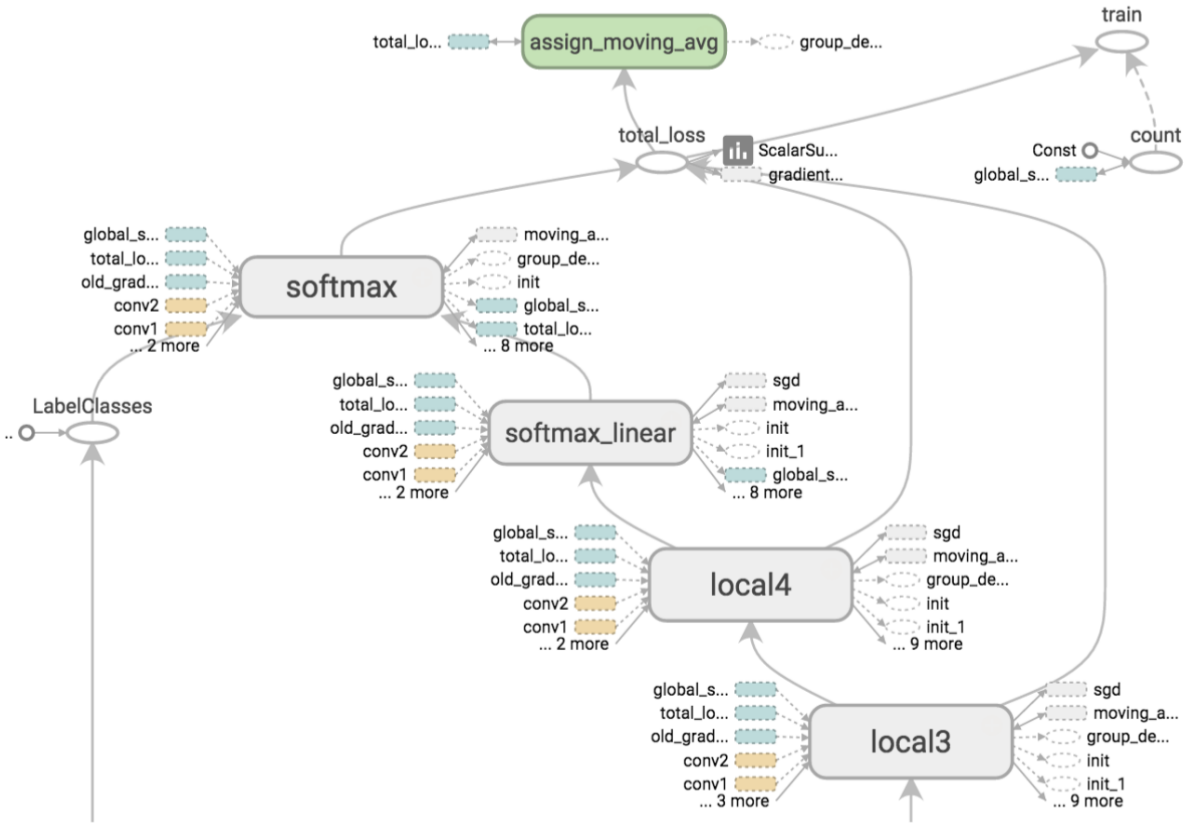

Abbildung 9: TensorBoard Visualisierung eines CNN [MA15]

3.13 Zukünftige Arbeit

Das Google Brain Team arbeitet noch immer an TensorFlow. Unter anderem sollen Teilgraphen vom Nutzer als eine wiederbenutzbare Komponente benutzt werden können und zukünftig sollen Funktionen, welche vom Nutzer z.B. in Python programmiert wurden, auch in C++ genutzt werden können.

4 Herausforderungen und Möglichkeiten bei der Anwendung in den Geowissenschaften [AK17]

Im Folgenden werden Herausforderungen und Möglichkeiten bei der Anwendung von Machine Learning Algorithmen in den Geowissenschaften betrachtet [AK17].

4.1 Einführung

Aktuelle Probleme unsere Gesellschaft betreffend sind oft geophysikalischer Natur

- Einfluss des Klimawandels
- Messung der Luftverschmutzung
- Gefahr durch Hurrikans, Erdbeben, Vulkanausbrüche
- Konsum von Wasser
- Nutzung von Ressourcen

Diese Probleme werden durch die Physik, Geologie, Geophysik, Biologie, Chemie, Hydrologie etc. erforscht. Durch verbesserte Sensoren und andere Technologien sind wir immer mehr in der Lage größere Datenmengen zu messen. Da Machine Learning große Datenmengen für einen Erfolg braucht, wird Machine Learning also immer attraktiver. Jedoch sind gemessene geowissenschaftliche Daten durch die zugrundeliegenden physikalen Gesetze oft komplex, wodurch sich andere Herausforderungen an typische Machine Learning Algorithmen stellen.

4.2 Quellen der Daten

Die Erde interagiert mit allen Disziplinen, wodurch eine hohe Komplexität entsteht. Im Grunde kann es zwei Quellen für geowissenschaftliche Daten geben: Beobachtete Daten, aufgenommen durch Sensoren und modellierte Daten basierend auf dem bekannten physikalischen Wissen (meist vereinfacht im Vergleich zu beobachteten Daten).
Beobachtete Daten werden durch viele unterschiedliche Unternehmen (NASA, ESA, Ölfirmen, Universitäten, Wetterdienste etc.) gemessen und teilweise der Öffentlichkeit zur Verfügung gestellt. Die Daten können zum Beispiel mit Satelitten, Sensoren in Flugzeugen, Dronen etc., in-situ Sensoren am Boden, in der Luft oder im Meer aufgenommen werden. Dabei wird die Auflösung und Genauigkeit, durch die Verbesserung der Technologien, immer besser. Es gibt zwei Datentypen, Punktdaten (von Flugzeugen, Schiffen etc.) oder Daten, welchen ein Raster zugrunde liegt (Satellitenbilder etc.). Die Datentypen können mit Interpolation ineinander umgerechnet werden.
Modellierte Daten unterliegen einer Interaktion verschiedener Forschungs-Disziplinen. Modelle werden basierend auf physikalischen Gesetzen erzeugt (Ozeanströmung z.B.),

aber oft gibt es keine analytische Lösung. Eine numerische Näherung wird daher angenommen, welche das physikalische Problem allerdings nie exakt wiederspiegelt. Noch dazu werden oft Vereinfachungen angenommen, um ein komplexes Problem lösbar zu machen.

4.3 Herausforderungen für die Geowissenschaften

Amorphe Ränder in Ort und Zeit sind oft eines der schwerwiegendsten Probleme. Dabei sind amorphe Ränder nicht so klar definiert wie Objekte in anderen Domänen, wie zum Beispiel eine Cola-Dose im Supermarktregal. Geowissenschaftliche Objekte bestehen meist aus Wellen und kohärenten Strukturen in allen Materien. Die Ränder sind dadurch viel komplexer, als in der diskreten Orts-Domäne.
Weiterhin sind Beobachtungen in den Geowissenschaften oft auto-korreliert in Raum und Zeit. Zum Beispiel ist ein Ort in der direkten Nachbarschaft eines Waldes meistens auch mit Wald bedeckt. Die Änderung eines Zustandes erfolgt dann meistens in der Zeit, wodurch eine höhere Abhängigkeit zu benachbarten Orten entsteht. Es sind keine unabhängigen Variablen mehr, wobei das eigentlich die Vorraussetzung für die meisten Algorithmen ist.
Die hohe Dimensionalität der Daten ist ein weiteres Problem. Da das Erd-System sehr komplex ist, gibt es eine hohe Anzahl an Variablen, welche sich gegenseitig beeinflussen. Es muss also eine Regression für sehr hochdimensionale Räume durchgeführt werden.
Noch dazu sind Heterogenitäten in Raum und Zeit ein Problem (z.B. jahreszeitliche Schwankungen bis hin zu sehr langskaligen geologischen Änderungen (Polaritätswechsel der Erde)). Es ist schwierig Machine Learning für alle Regionen auf einmal anzuwenden, es müssen also regionale Modelle trainiert werden. Ein Ansatz hierfür ist Multi-Task Learning, wobei Teilgebiete alleine angelernt werden um sie danach zusammenzusetzen.
Ereignisse, welche von starkem Interesse sind, sind oft sehr selten (z.B. Vulkanausbrüche, starke Erdbeben etc.). Dadurch fehlen für das Training Datenmengen und es treten ungleiche Datenverteilungen für die Klassen auf. Für die meisten Machine Learning Algorithmen ist das ein Problem.
Oft sind Trainingsdaten mit unterschiedlichen Auflösungen vorhanden, wodurch ein Training ebenfalls erschwert wird. Unterschiedliche Auflösungen können zum Beispiel auftreten, wenn Daten über mehrere Jahrzehnte gemessen werden und sich dabei die Auflösung stetig verbessert. Man braucht Algorithmen, welche nicht automatisch upsampling durchführen, da nicht automatisch auf die höchste Auflösung geupsamplet werden darf. Es würden sonst Fehler in den Daten entstehen.
Ein ähnliches Problem ist Noise oder inkomplette Daten zum Beispiel durch den Ausfall eines Messsensors.
Mit transfer-learning kann ein bisher trainiertes Modell an einer komplett neuen Aufgabe angewendet werden. Dadurch kann man auch mit kleineren Datenmengen trainieren, um das Problem einer zu kleinen Datenmenge etwas zu umgehen. Kleine Datenmengen sind oft durch eine historische Datensammlung bedingt (Möglichkeiten zu messen waren z.B. erst ab 1970 vorhanden).

Eines der Hauptprobleme mit supervised Learning ist das Labeln, welches durch einen Menschen durchgeführt werden muss. Dabei entstehen oft Fehler, oder Ungenauigkeiten, welche das Training verschlechtern. Aber auch die physikalischen Annahmen bei der Nutzung synthetischer Daten sind ungenauer als die beobachteten Daten. Dadurch entsteht immer ein Mangel an der Grundwahrheit. Der Machine Learning Algorithmus lernt dann auf einer leicht fehlerhaften Datenmenge. Dieses Problem kann man mit unsupervised Learning etwas umgehen.

4.4 Zusammenfassung

Deep Neural Networks sind eine gute Wahl für geowissenschaftliche Daten, da sie auch komplexere Zusammenhänge finden können. Da die Ergebnisse am Ende auch interpretierbar sein sollten, sollten ebenso theorie-basierte Daten erzeugt und herangezogen werden. Nicht nur beobachtete Daten.

5 Beispielhafte Anwendung während meiner Masterarbeit

Ich versuche ein Problem in der Geophysik mit einem Neuronalen Netzwerk zu lösen. Hier wird ein einfaches Beispiel dazu gezeigt, welches bisher funktioniert hat.

In Abbildung 10 sieht man eine seismische Akquisition mit den Quellen S1 und S2, den Geophonen G(1) bis G(6) und dem Diffraktor D. Die eingezeichneten Linien sind die Strahlwege.

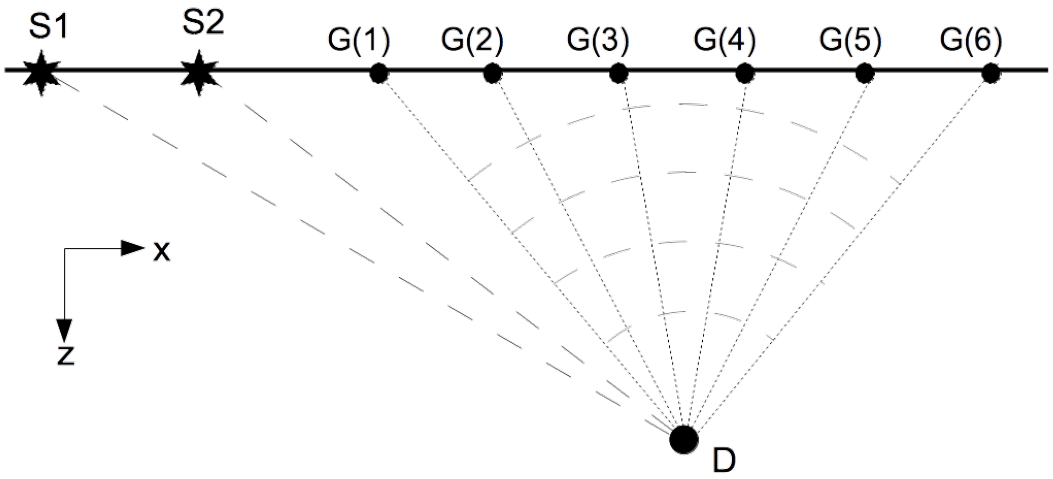

Abbildung 10: Skizze Akquisition mit Diffraktor D

Die Quelle sendet ein Signal aus, welches an einem Impedanzkontrast im Untergrund reflektiert oder diffraktiert wird. Wir können in der angewandten Seismik unter anderem zwei Arten von Events beobachten. Die Reflexion und die Diffraktion. In Ab-

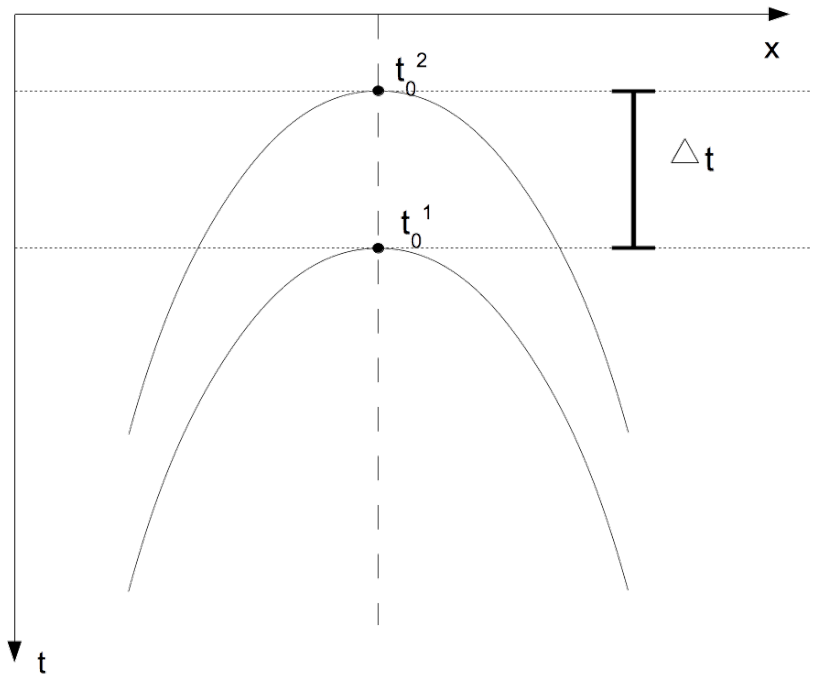

Abbildung 11: Skizze der Diffraktionen zweier Shotgather mit identischem Diffraktor

bildung 11 sind zwei Diffraktionen dargestellt. Beide kommen von unterschiedlichen Shotgathern (unterschiedliche Quellposition). Der Diffraktionspunkt im Untergrund ist bei beiden identisch, wodurch sich die Form der Diffraktion (Hyperbel im homogenen und isotropen Fall) nicht ändert. Diese Eigenschaft lässt sich ausnutzen, um eine Diffraktionsverstärkung zu erreichen. Die identische Form der Diffraktionen führt mit einer Kreuzkorrelation in Offset-Richtung (x-Richtung) zu einem geraden Event für Diffraktionen. Für Reflexionen ist durch die unterschiedliche Form mit einem gebogenen Event zu rechnen. Das Ziel ist, die geraden Events zu detektieren und die Information in weiteren Schritten danach zu nutzen, Diffraktionen zu verstärken. Da im Folgenden nur ein sehr einfaches Modell angenommen wird, mit jeweils einer Diffraktion und einer Reflexion, sei hier nur erwähnt, dass seismische Sektionen aus der Überlagerung hunderter und tausender Diffraktionen und Reflexionen bestehen. Dadurch wird eine viel höhere Komplexität erreicht und der im Folgenden verwendete Graph würde keinesfalls ausreichen. Es sei somit nur auf einer Beispiel zur Anwendung eines neuronalen Netzes verwiesen.
Bei einer Detektion eines solchen geraden Events, könnte man die Zeitdifferenz beider zugehörigen Diffraktionen errechnen. Diese Information würde dazu dienen, Diffraktionen zu verstärken und alle anderen Events relativ dazu abzuschwächen. Ein erster Versuch bei der Detektion einer Zeitdifferenz ist die Unterscheidung zwischen geraden und gebogenen Events. Dazu wurde ein einfaches neuronales Netzwerk verwendet, mit zwei Schichten. Der Datensatz besteht aus reinen Diffraktionen (Abbildung 12) und aus reinen Reflexionen (Abbildung 13).

In Abbildung 15 findet sich die Kreuzkorrelation zweier Diffraktionen und in Ab-

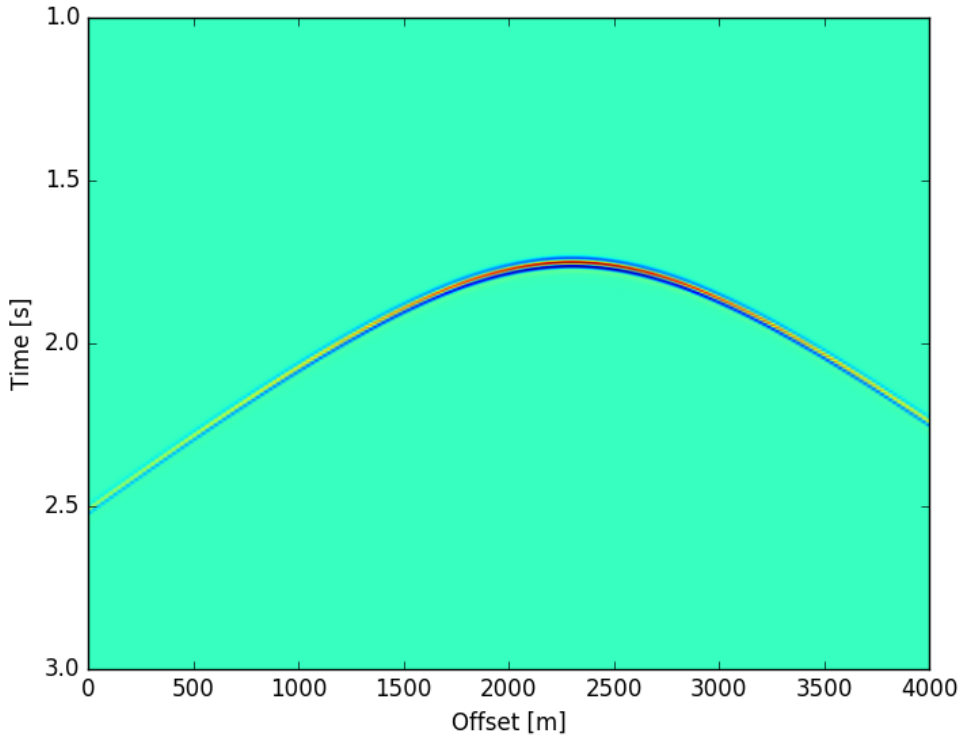

Abbildung 12: Diffraktion in der reflexionsseismischen Sektion

bildung 16 zweier Reflexionen. Hier sind die geraden und gebogenen Events zu sehen. Eine Skizze der Kreuzkorrelationen findet sich auch in Abbildung 14, wobei die rote Kreuzkorrelation zu den Reflexionen gehört und blau zu den Diffraktionen.

Der Datensatz besteht aus 800 Seismogrammen, 700 davon wurden zum Trainieren benutzt und 100 zur Validation. Die Daten wurden selbst gelabelt und mit supervised Learning trainiert. Der verwendete Graph findet sich in Abbildung 17. Es ist ein simpler zweischichten Graph mit nur einem Input-Layer, einem Hidden-Layer und einem Output-Layer. Als Aktivierungsfunktion wurde die ReLu-Funktion gewählt (Abbildung 18). Hier wurde eine Lineare Regression durchgeführt (Abbildung 18), welche dann nicht-lineare Anteile bekommt. Die Darstellung des Graphen erfolgte mit TensorBoard.

Für das Layer 1 und 2 wurden jeweils die Variablen und der Bias für jede Iteration in Form eines Histogramms dargestellt. Die Gewichte wurden initial mit einer Gaußschen Verteilung angenommen und der Bias war mit einem Peak bei 0.1 verteilt. Siehe Abbildung 19 und 20.

Hier ist zu erkennen, dass in dem Layer 1 keine Änderung der Gewichte stattfindet, was darauf schließen lässt, dass hier nicht trainiert wurde. Auch beim Bias ist zu erkennen, dass er gleichverteilt wird und somit nicht trainiert wird. In Layer 2 hingegen zeigt die Verteilung der Gewichte, sowie des Bias einen klaren Trend mit 2 Maxima

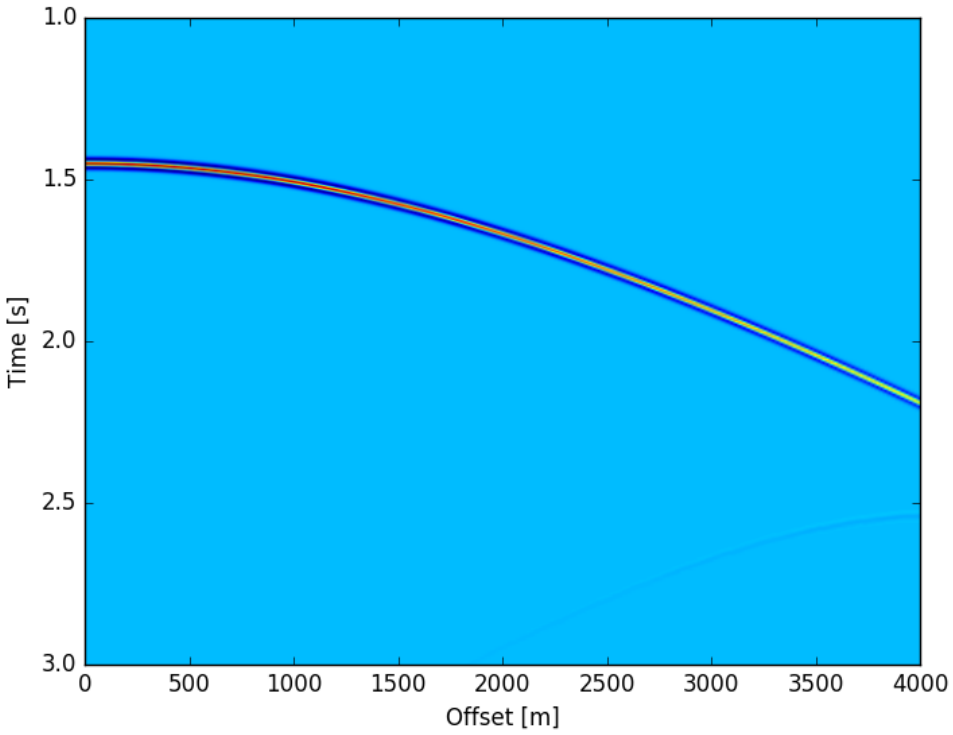

Abbildung 13: Reflexion in der reflexionsseismischen Sektion

(siehe Abbildung 21 und 22). Die zwei Maxima könnten für das gerade Event und das gebogene Event stehen. Eine klare Aussage darüber ist allerdings nicht möglich.

Die Loss-Function (Cross-Entropy) tendiert nach der 10. Iteration gegen null (siehe Abbildung 23) und die Accuracy erreicht nach der 17. Iteration den Wert 1 (siehe Abbildung 24). Der Graph ist also perfekt trainiert und ordnet gerade und gebogene Events zu 100 Prozent zu. Auch wenn das erste Layer nicht richtig funktioniert, dient es vielleicht als Filter. Der Graph trainiert ohne dieses erste Layer nämlich langsamer und braucht ca. 70 Iterationen bis zu einer Accuracy von 1.

Für das weitere Vorgehen möchte ich die Zeitdifferenz herausfinden. Dazu wäre es hilfreich eine Heatmap mit einer pixelweisen Wahrscheinlichkeit eines geraden Events zu erhalten, wie in Abbildung 25 skizziert. Über das Finden des Maximums kann die Zeitdifferenz abgelesen werden. Um solch eine Heatmap zu erhalten, muss ein Segmentationsproblem gelöst werden, welches die Kreuzkorrelation pixelweise in verschiedene Klassen unterteilt. Dafür werden zwei Autoencoder (siehe Abbildung 26) verwendet, welche miteinander verknüpft sind. Sie bilden ein sogenanntes W-Net und sind dem Bereich unsupervised Learning zuzuordnen. Zuerst findet ein Downsampling über Convolutional-Layers und sogenanntes Max-pooling statt. Wir erhalten zwei Klassen (gebogen / gerade). Danach wird die Kreuzkorrelation aus den zwei Klassen wieder aufgebaut und wir erhalten eine pixelweise Wahrscheinlichkeit für ein gebogenes oder gerades Event. Wendet man die gleiche Vorgehensweise ein zweites mal an, kann das ursprüngliche Input-Bild rekonstruiert werden. Der Unterschied

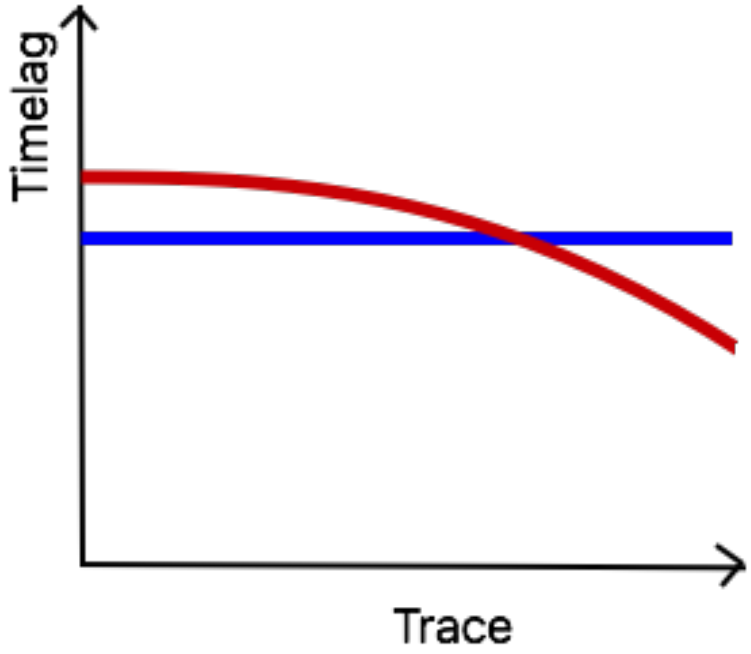

Abbildung 14: Skizze der Kreuzkorrelation zweier Diffraktionen und zweier Reflexionen

zwischen original Bild und der Rekonstruktion soll minimiert werden. So kann das Netz zusätzlich trainiert werden. Bei einer erfolgreichen Anwendung ist die direkte Anwendung auf unprozessierte seismische Daten geplant.

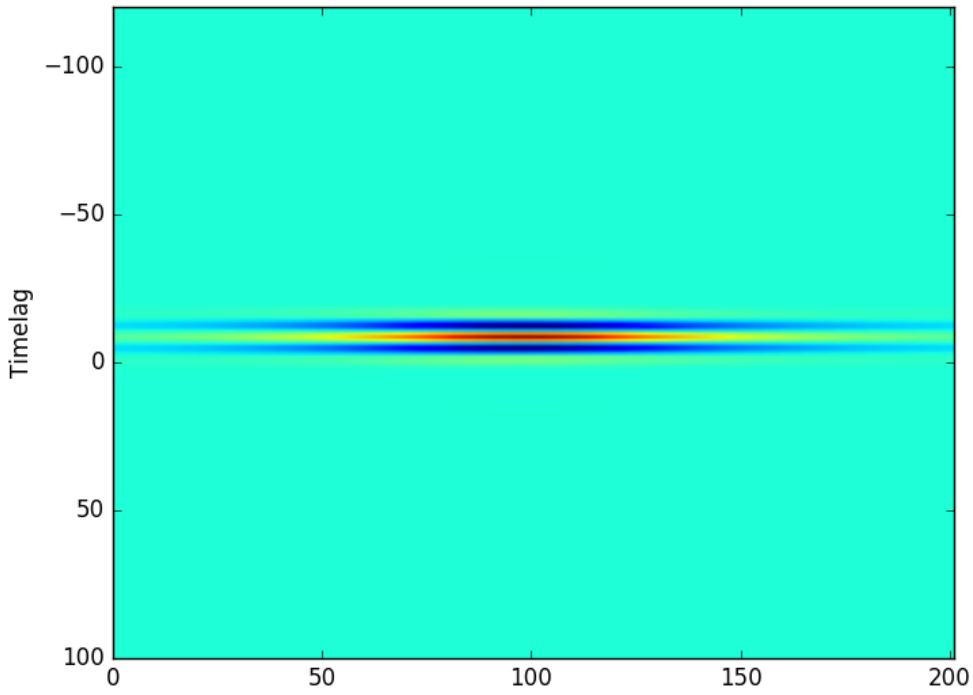

Abbildung 15: Kreuzkorrelation zweier Diffraktionen wie in Abbildung 12

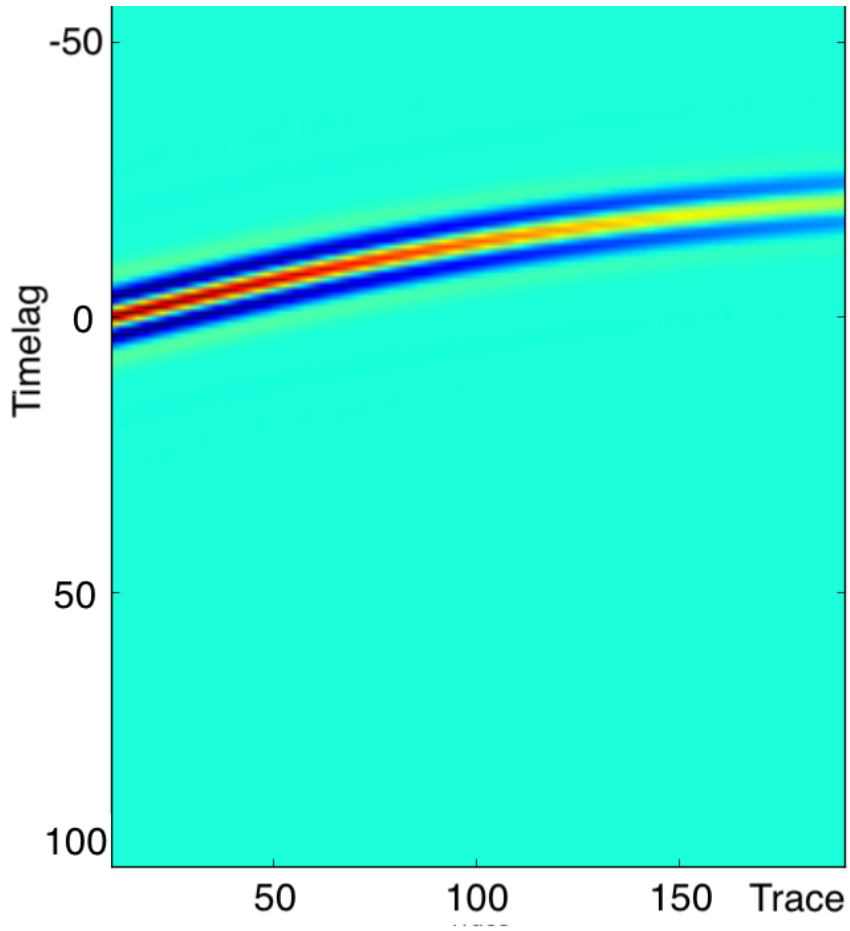

Abbildung 16: Kreuzkorrelation zweier Reflexionen wie in Abbildung 13

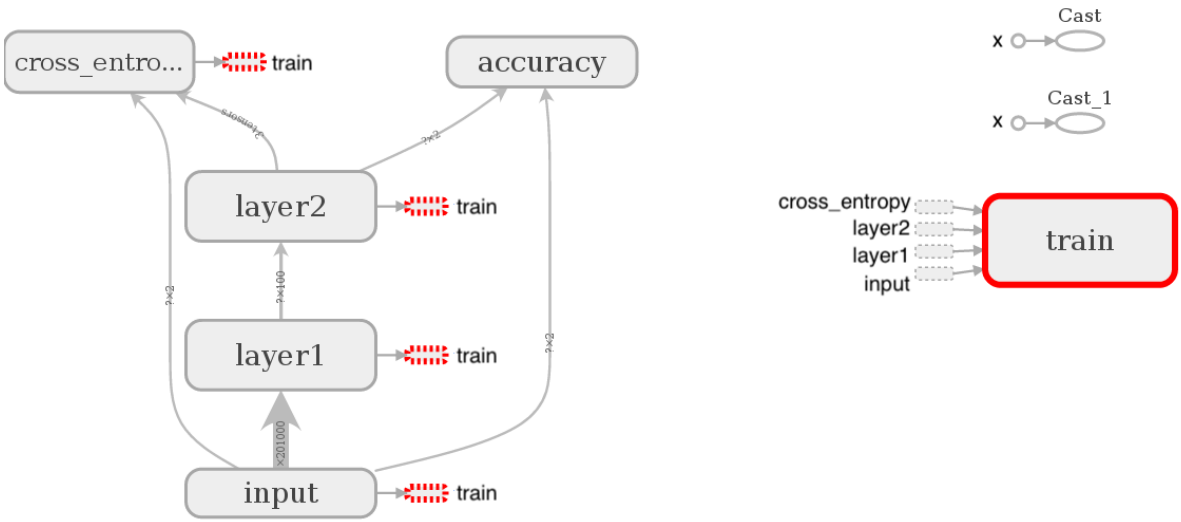

Abbildung 17: Visualisierung des benutzten Graphens mit TensorBoard

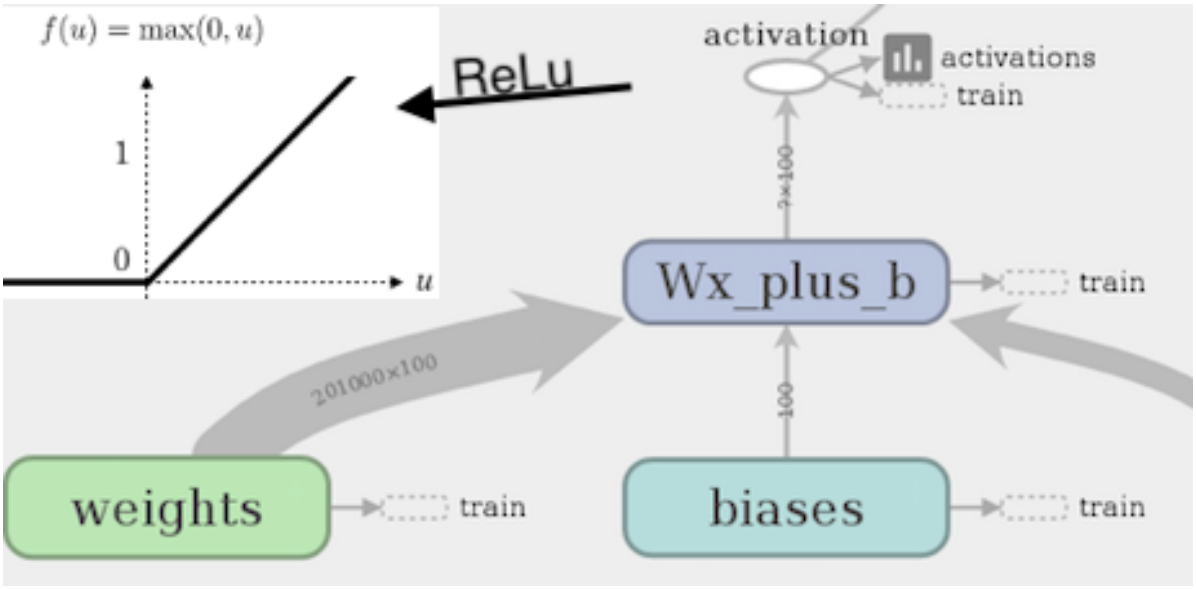

Abbildung 18: Visualisierung der ersten Schicht mit TensorBoard

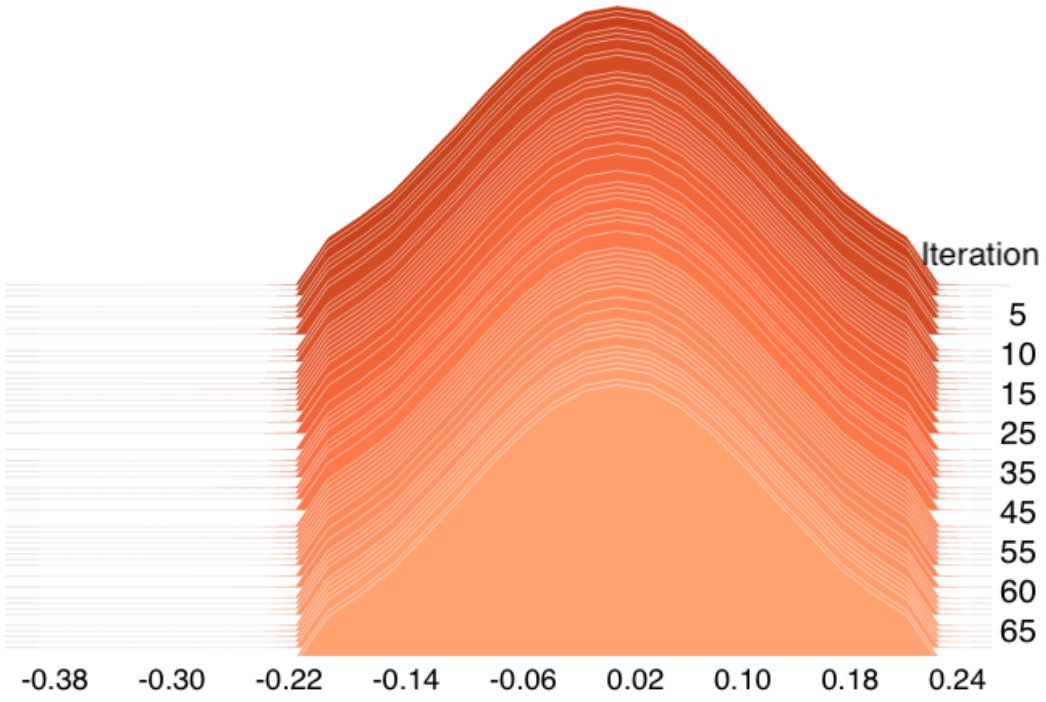

Abbildung 19: Histogramm der Gewichte in Schicht 1 (mit TensorBoard)

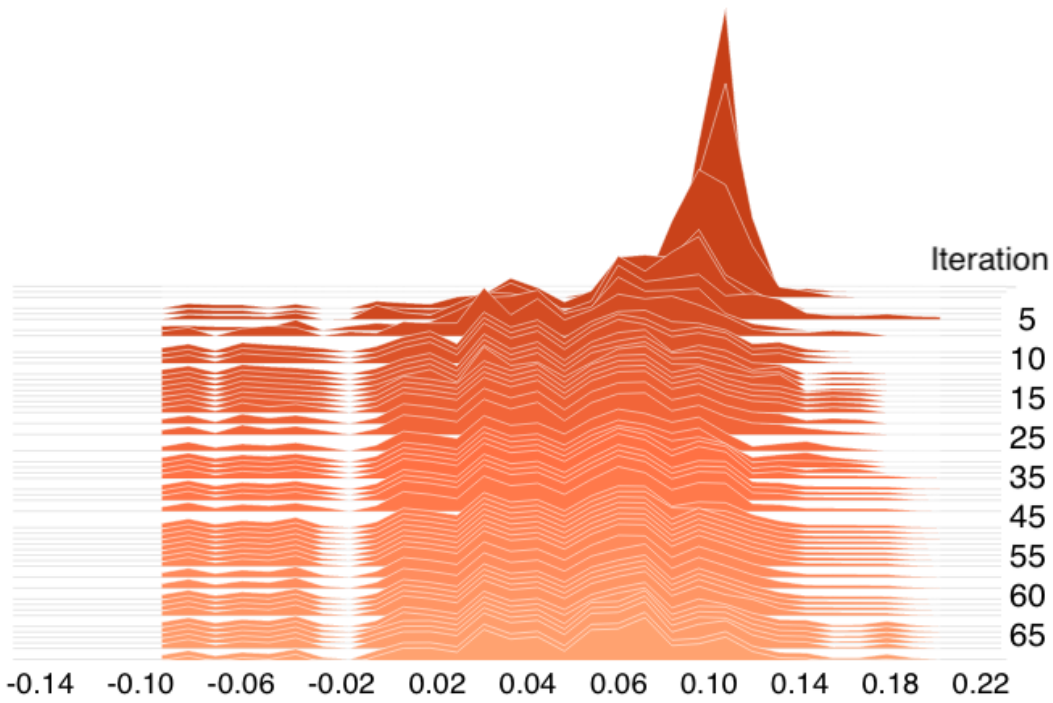

Abbildung 20: Histogramm des Bias in Schicht 1 (mit TensorBoard)

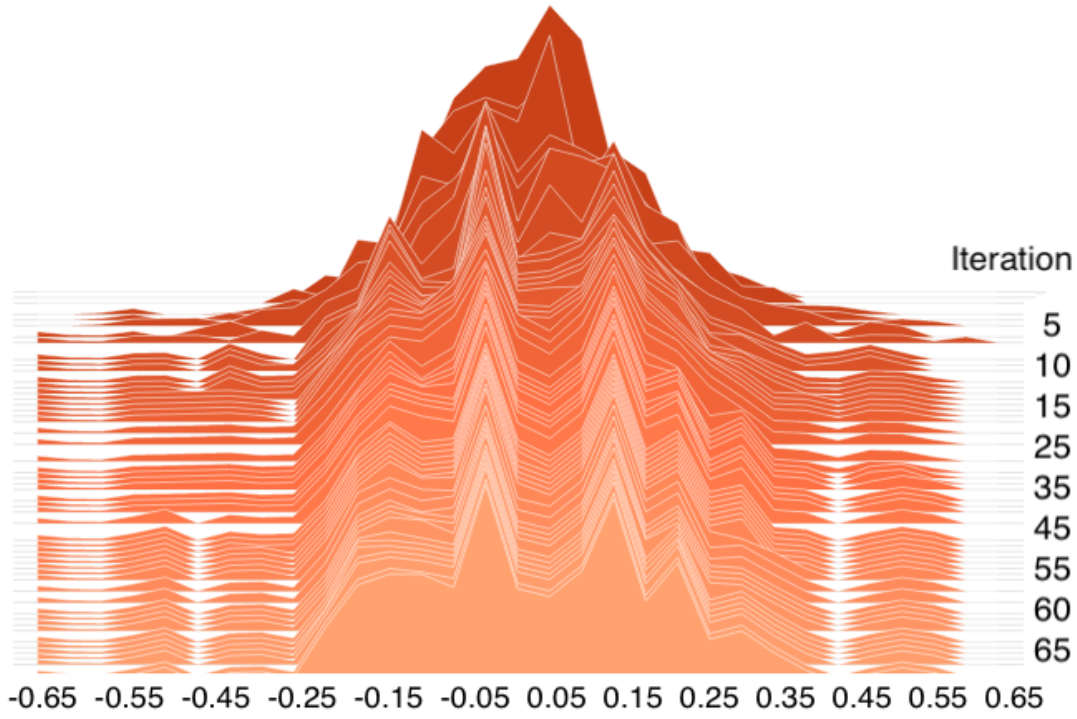

Abbildung 21: Histogramm der Gewichte in Schicht 2 (mit TensorBoard)

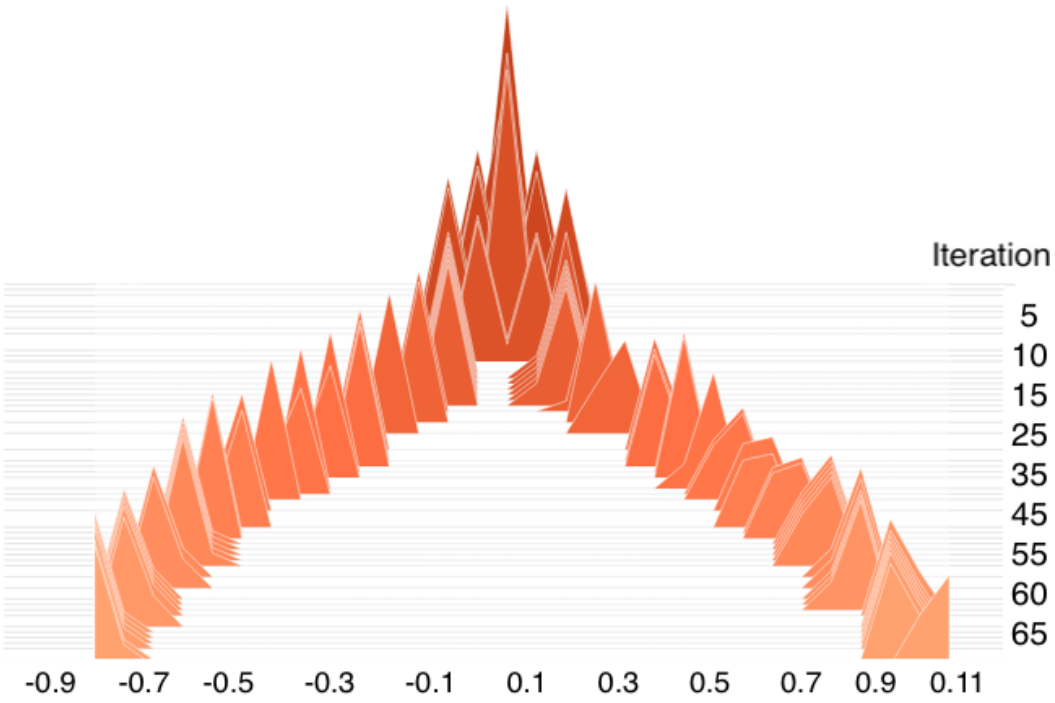

Abbildung 22: Histogramm des Bias in Schicht 2 (mit TensorBoard)

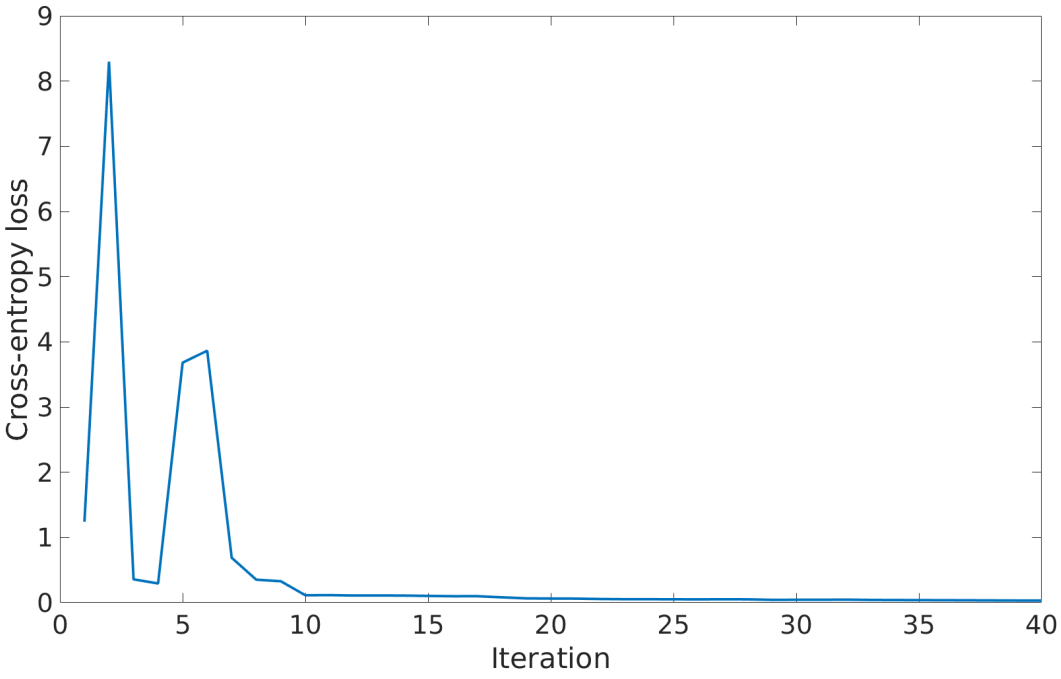

Abbildung 23: Verlustfunktion Cross-Entropy visualisiert mit TensorBoard

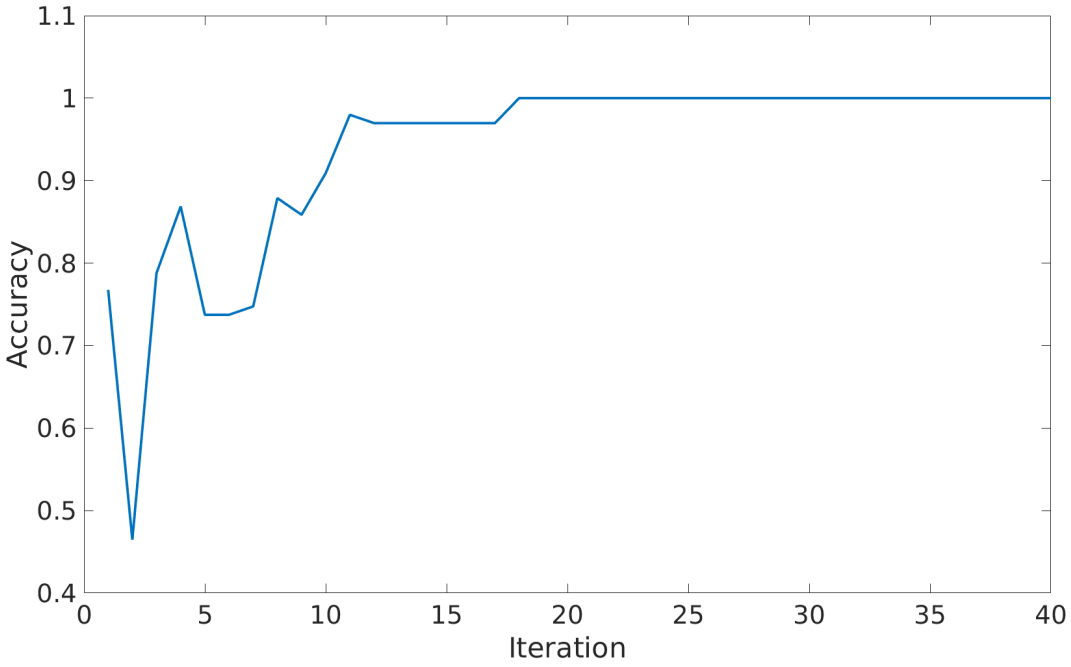

Abbildung 24: Accuracy visualisiert mit TensorBoard

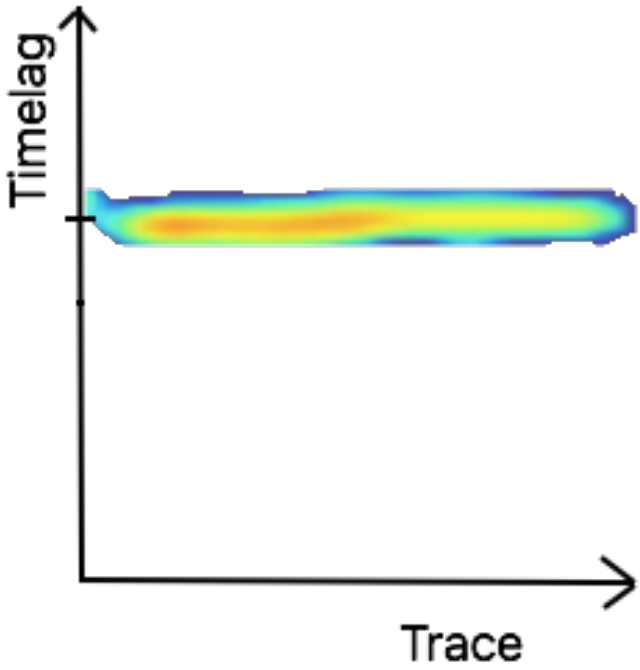

Abbildung 25: Beispiel einer Heatmap

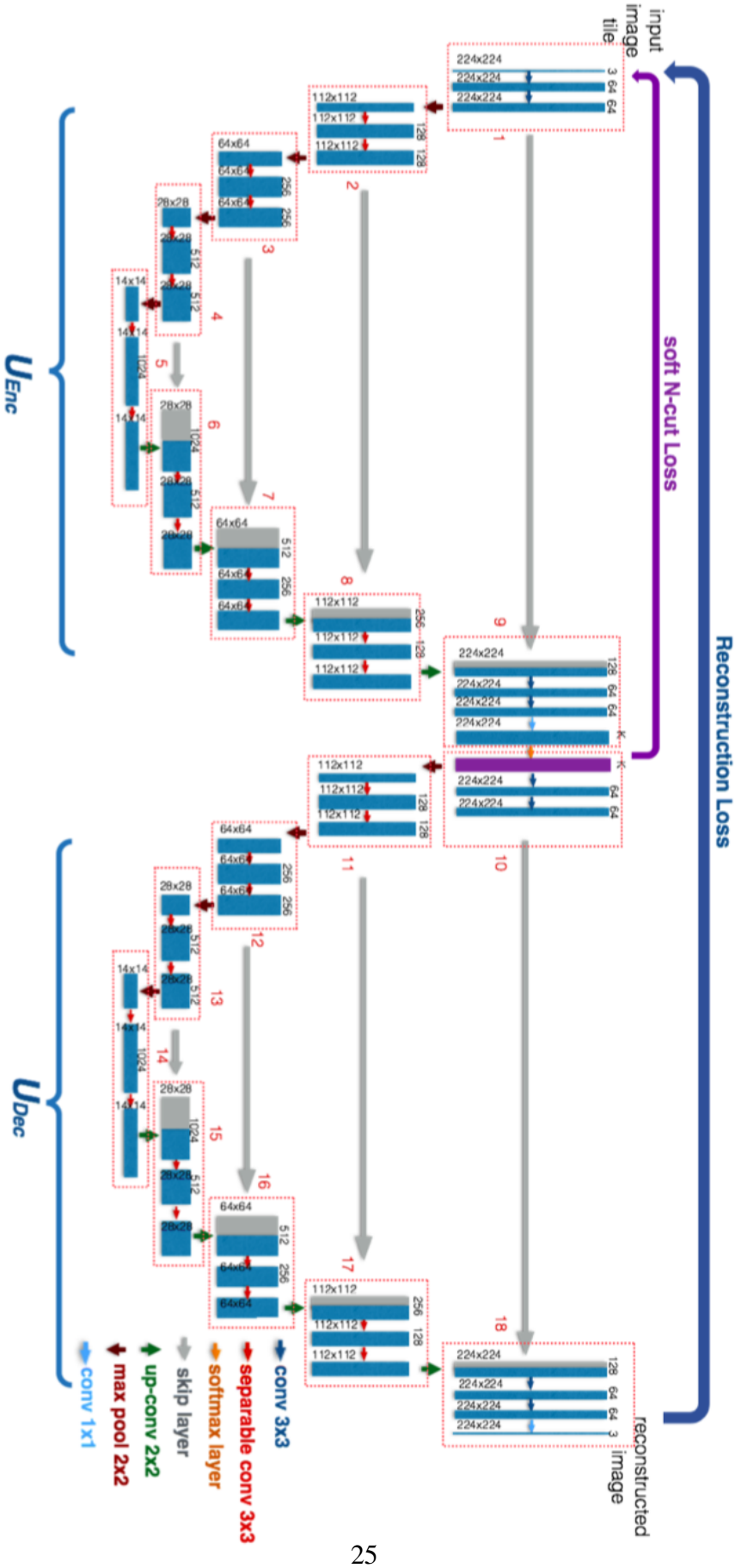

Abbildung 26: Autoencoder für weitere Schritte [XX17]

Literatur

[AK17] A. KARPATNE, S. Ravela et a. I. Ebert-Uphoff: *Machine Learning for the Geosciences: Challenges and Opportunities*. 2017

[Gra16] GRACHTEN, Maarten: *A very brief overview of deep learning*. 2016

[MA15] M. ABADI, P. Barham et a. A. Agarwal A. A. Agarwal: *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. 2015

[XX17] XIDE XIA, Brian K.: *W-Net: A Deep Model for Fully Unsupervised Image Segmentation*. 2017